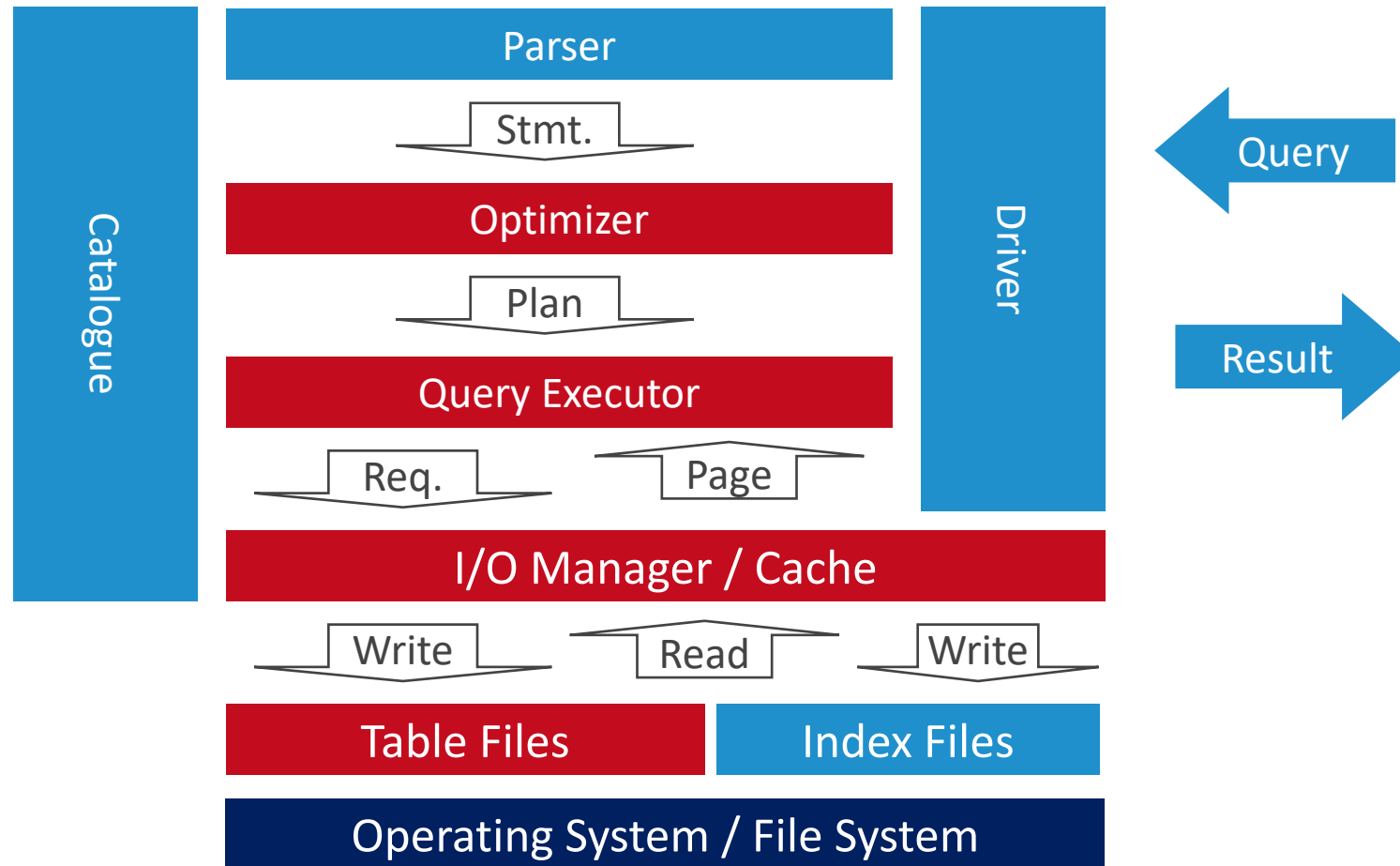


DBTLAB – Exercise 1: Table Page Layout

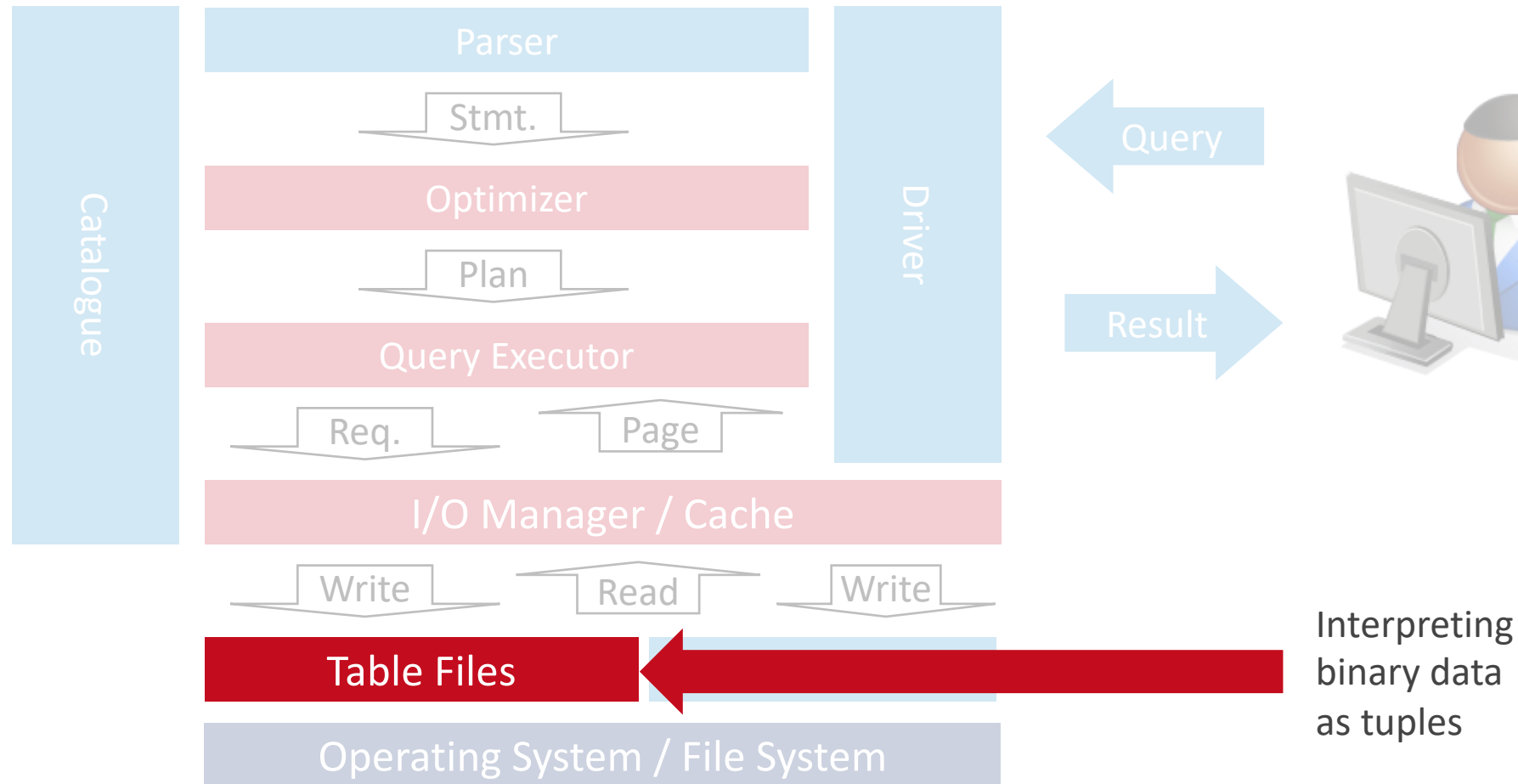
Viktor Rosenfeld

based on material from Volker Markl, Alexander Alexandrov, Jonas Traub, Martin Kiefer

Basic system architecture



Where are we today?

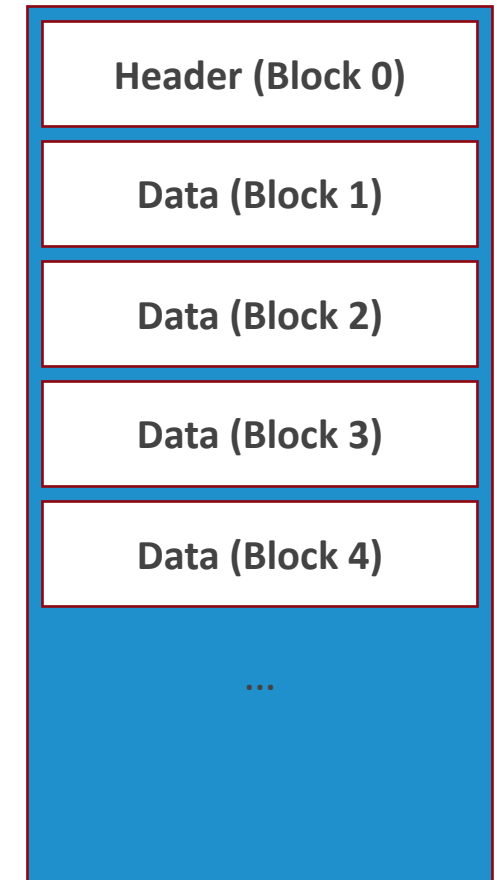


Background: Block-based data access

- Hard drives and SSDs are accessed in blocks.
- Terminology
 - *Block* is usually a unit of physical storage (organized by the disk controller).
 - *Page* is usually a unit of memory (organized by the OS).
 - *Page* and *Block* used as synonyms here!
- Pages/Blocks are units of I/O.
- Every I/O request reads/writes exactly one block.
- I/O requests can later be chained to process consecutive blocks.

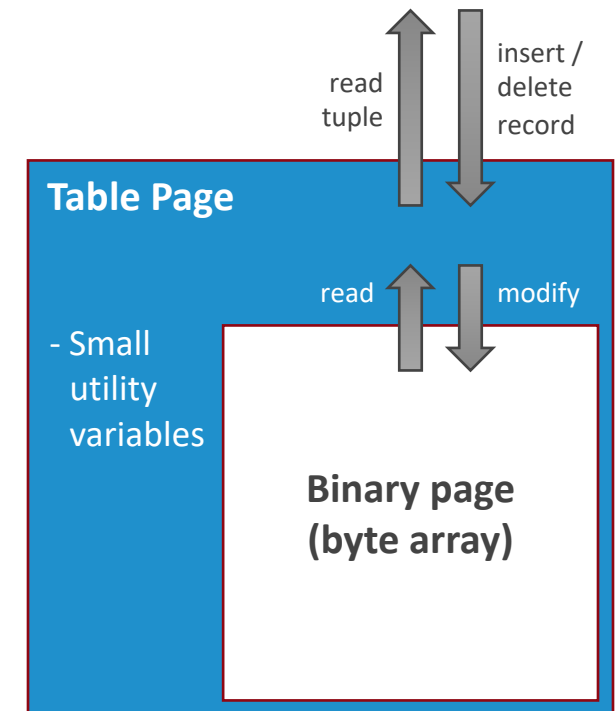
Table files

- Tables are stored as table files.
- A table file is a sequence of blocks.
- First block of table is the header.
 - Contains schema description of the table.
 - Only looked at during first access (DBMS startup) to read metadata.
- Subsequent blocks are data pages.
 - Table organized as heap file, no logical order of the records.
 - No schema information on a data page.



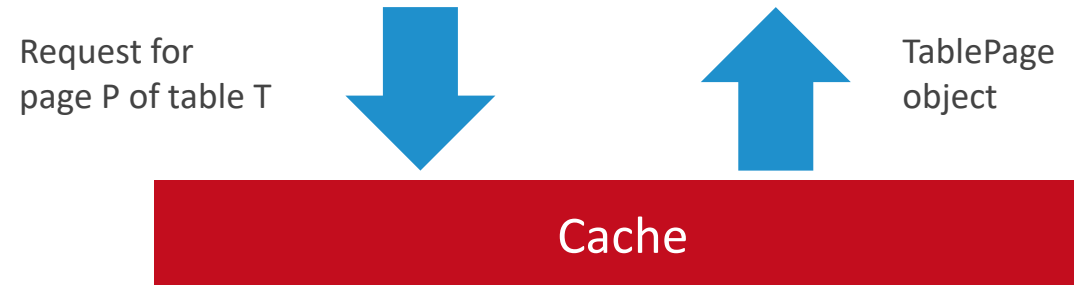
TablePage objects

- A block is read from disk – the application gets a byte buffer (byte[]).
 - Binary sequence, no structure, yet.
- The *TablePage* Object wraps the byte buffer and makes records accessible.
 - Operations like *getDataTuple()*, *insertTuple()*, ...
- TablePage is **not a copy** of the binary page.
 - Every modification to the TablePage objects directly modifies the wrapped byte buffer.
 - **Never copy arrays or array contents, or create new arrays!**
 - Only serialize and deserialize data (explained later).
- TablePage contains some additional information
 - Modification flag, expiration flag



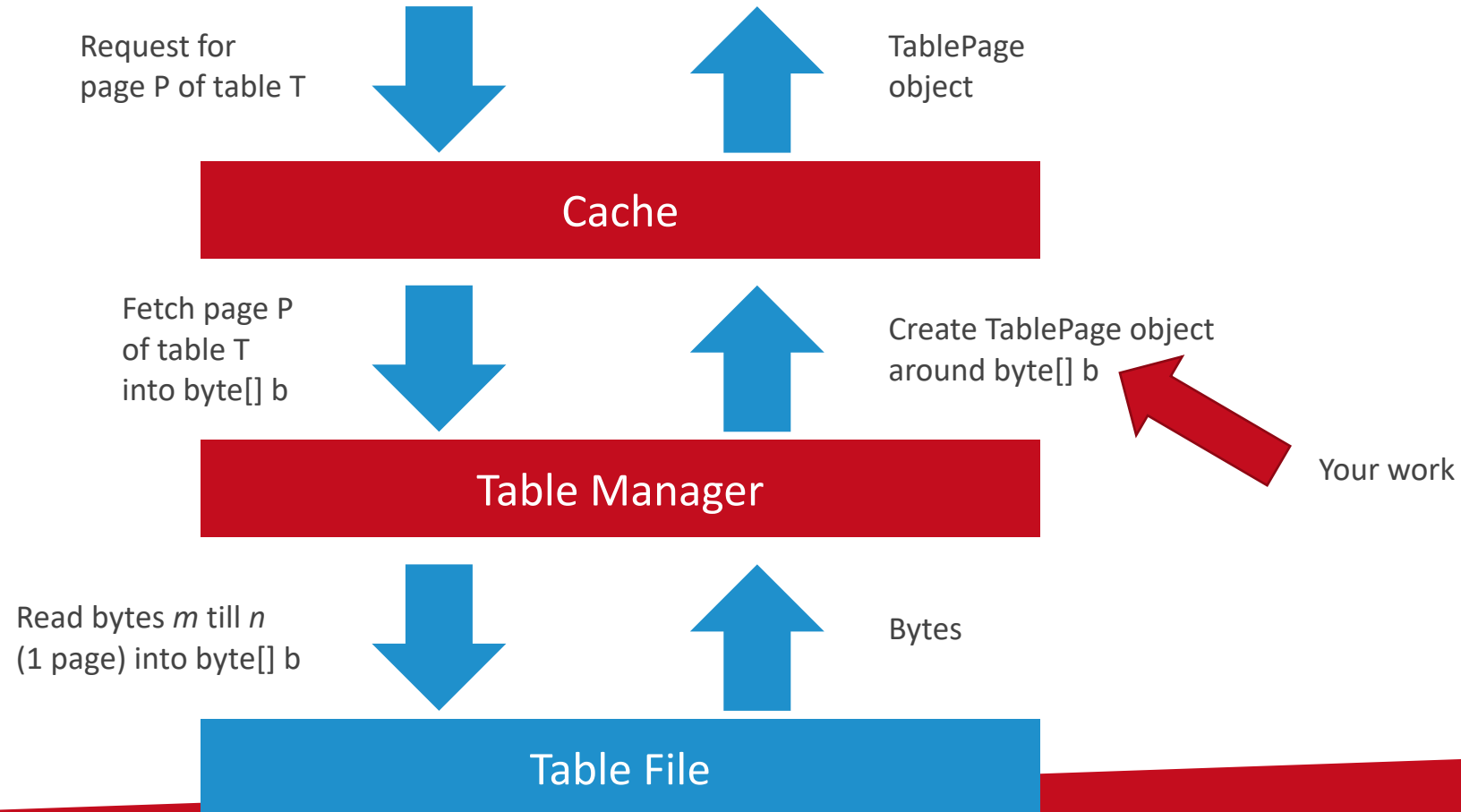
How TablePages are used in the system

- Case 1: Access an existing TablePage which is already loaded in the cache.



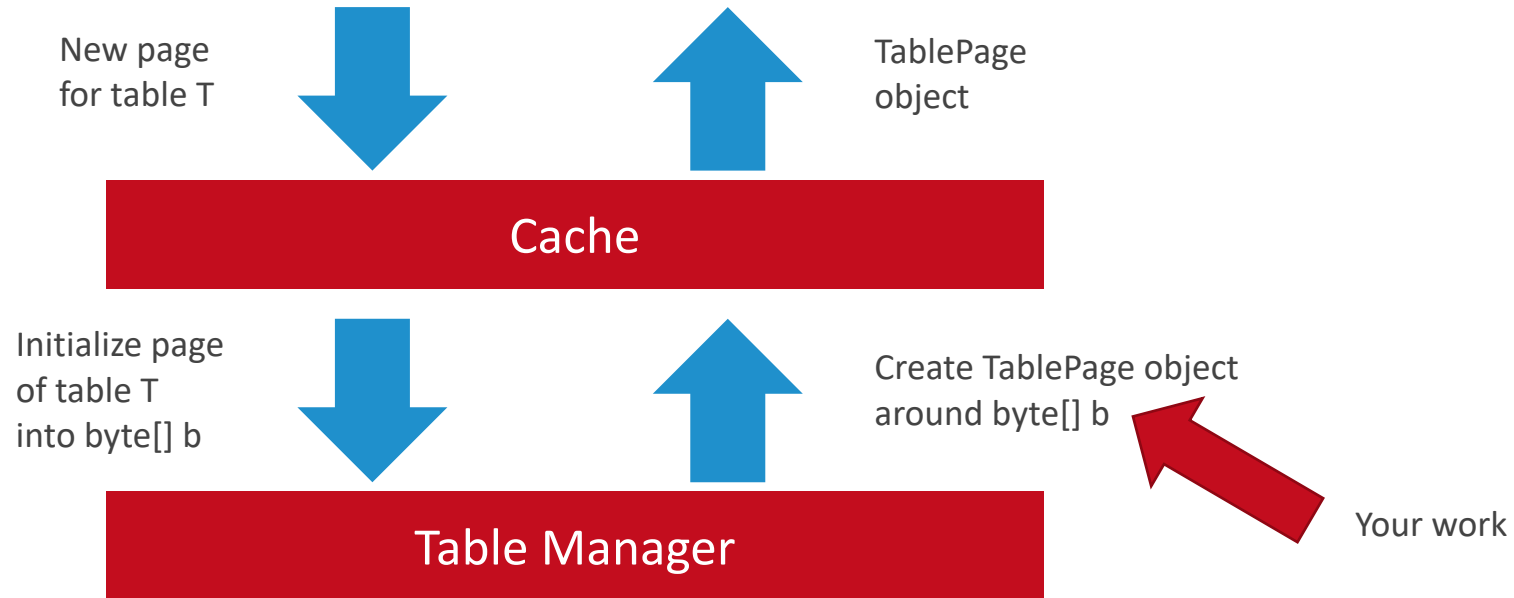
How TablePages are used in the system

- Case 2: Access an existing TablePage which is not in the cache.



How TablePages are used in the system

- Case 3: Initialize a new page.



TablePage API

TablePage API

- Methods that access metadata of the TablePage object
- Methods that access metadata about the contents
- Methods to insert/delete/retrieve a single tuple
- Methods to iterate over all the tuples in a page.

Access TablePage metadata

- Do not read/modify the wrapped byte array.
- `boolean hasBeenModified()`
 - Has the page been modified since it was read from disk?
 - If so, it must be written back when it is evicted from the cache.
- `void markExpired()`
 - Called when a page is evicted from the cache.
 - The byte array might now belong to another page!
- `boolean isExpired()`

Access metadata about the contents

- Information must always be in sync with the wrapped byte array!
 - Tip: Read answer directly from the wrapped byte array.
- `int getPageNumber()`
 - Every page in a table file is numbered.
- `int getNumRecordsOnPage()`
 - How many tuples are stored on that page?

Insert/delete/retrieve a single tuple

- Again: Information must always be in sync with the wrapped byte array!
- `boolean insertTuple(DataTuple tuple)`
- `void deleteTuple(int position)`
- `DataTuple getDataTuple(int position, long columnBitmap, int numCols)`
 - Return (a projection of) a stored tuple.
- `DataTuple getDataTuple(LowLevelPredicate[] preds, int position, long columnBitmap, int numCols)`
 - Return (a projection of) a stored tuple if it passes a set of predicates.

Iterate over all the tuples in a page

- TupleIterator `getIterator(int numCols, long columnBitmap)`
 - Iterate over (a projection of) all stored tuples in the page.
- TupleIterator `getIterator(LowLevelPredicate[] preds, int numCols, long columnBitmap)`
 - Iterate over (a projection of) all stored tuples that pass a set of predicates.
- TupleRIDIterator `getIteratorWithRID()`
 - Iterates over all stored tuples in the page and returns them together with their RID.

Projection

- Retrieve a subset of the tuple attributes.
 - Encoded by a column bitmap.
 - Least significant bit represents first attribute in the tuple.

- Stored tuple:

Name	Department	Salary
------	------------	--------

- Column bitmap = 110, Number of columns = 2

- Returned tuple

Department	Salary
------------	--------

Selection

- Retrieve only those tuples which pass a set of predicates.
 - *LowLevelPredicate* objects.
- **int** getColumnIndex()
 - On which column/attribute the predicate should be evaluated.
- **boolean** evaluate(DataField field)
 - Evaluate the predicate on the deserialized attribute.
- Predicates are evaluated before projection.
 - Extract fields that a predicate is defined on and evaluate.
 - If all fields pass, extract those that are required by the projection.

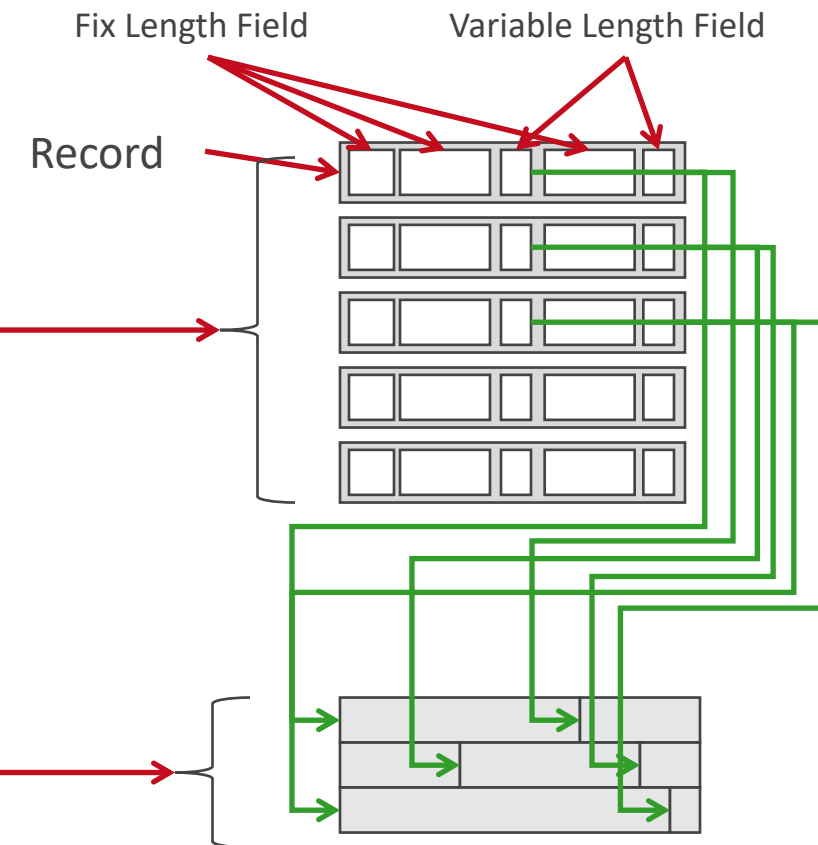
Data layout

Background: Table data layouts

- Row layout
 - All the attributes of a single tuple are stored together.
 - Good for transactional workloads (point queries, insertions, deletions, updates).
- Column layout
 - All the values of a single attribute/column are stored together.
 - Good for analytical workloads (aggregations).
- Hybrid layouts for mixed workloads.
- Fractured mirrors: Store both row and column layout.
- Vertical partitioning: Store some attributes in row layout and other attributes in column layout.
- PAX: A block contains all the attributes of a set of rows but they are stored in column layout.

TablePage row layout

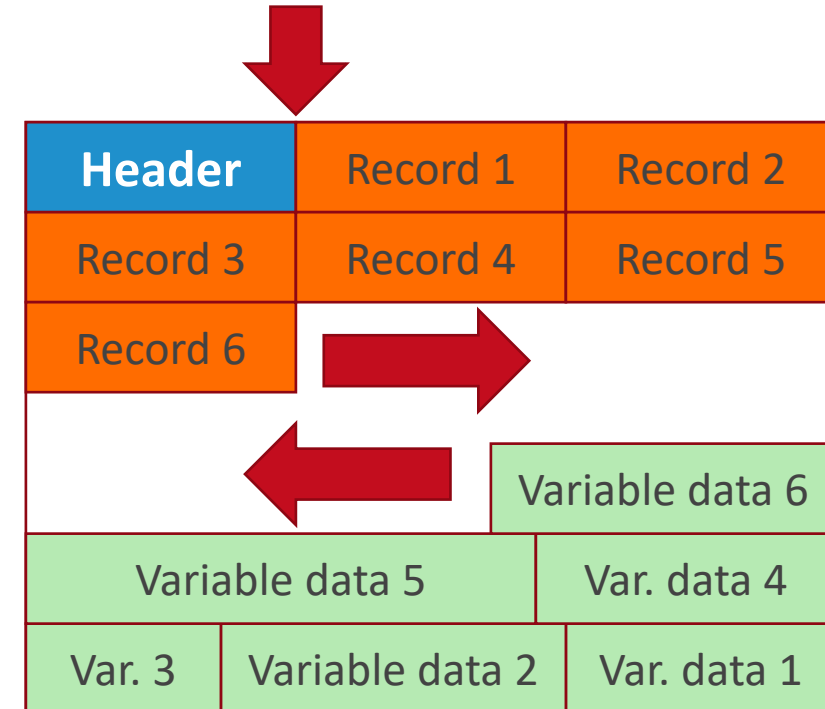
- Records are of fixed length.
 - Efficient random access based on record number (*position*).
- Variable length fields are stored in a dedicated chunk.
 - Referenced through pointer and length.
- Consequently, the block has three parts:
 - Header
 - Record Sequence
 - Variable Length Chunk



Layout inside the page

- Records for tuples are inserted from the beginning (after the header).
- The referenced variable length fields are inserted from the end.
 - Offset to variable-length chunk is current beginning and gets smaller after each insertion.
- Page grows from both sides, meets in the middle.

Start writing records after header



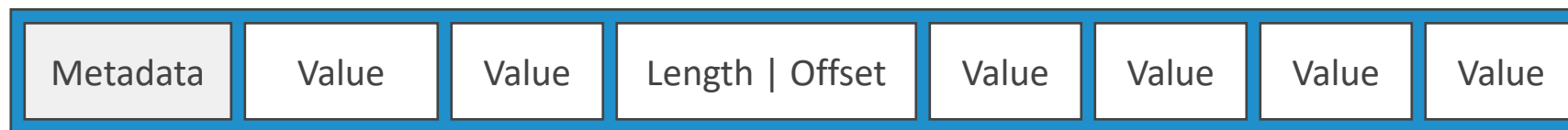
Start writing variable data at end

TablePage header

- 32 bytes in at the beginning of each page
 - `TablePage.TABLE_DATA_PAGE_HEADER_BYTES`
- 4 bytes magic number
 - Used to identify a page as a page table.
 - Fixed 32-bit value: `0xDEADBEEF`
 - `TablePage.TABLE_DATA_PAGE_HEADER_MAGIC_NUMBER`
- 4 bytes page number
 - Makes the page more self-contained
- 4 bytes number of records on the page
- 4 bytes record width
- 4 bytes offset of the start of the variable-length chunk
- Remainder is reserved

Record representation

- Record consists of Metadata, Values and Variable-Length-Field-Pointers.
- Metadata (4 bytes IntField): Per tuple information.
 - Tombstone: Least significant bit = 0/1 in the integer marks tuples as alive/deleted.
- Values are stored *as they are*, no compression here, for simplicity.
- Variable-Length-Field-Pointers are stored as BigIntField (8 bytes).
 - Lower 4 byte are an offset into the variable-length chunk.
 - Higher 4 byte are the length of the field.



Schema information

- TablePage object is always constructed with a *Schema*.
 - Number and type of attributes
 - Allows us to derive fixed size of a record.
- Classes inheriting from *DataField*: *IntField*, *VarcharField*, ...
 - Number of bytes
 - If it is an fixed length type or an array type.
 - Serialization methods: *encodeBinary()*
- *BasicType*
 - Enumerates all possible types in the system
- *DataType*
 - Enumerates all possible types in the system with additional properties.
 - Serialization and deserialization methods: *encodeBinary()*, *getFromBinary()*.

Serialization and deserialization

- Data has to be transformed from Java objects (subclasses of DataField) to byte array contents.
 - And back.
- A DataField subclass knows how to serialize itself.
 - `int encodeBinary(byte[] buffer, int offset)`
- Deserialization is handled through the DataType class which can be constructed from a BasicType.
- DataField `getFromBinary(byte[] binaryEncoded, int offset)`
 - For a fixed length data types (all except VARCHAR).
- DataField `getFromBinary(byte[] binaryEncoded, int offset, int len)`
 - For variable length data types.

Handling of NULL values

- NULL represented as a certain value in the domain of the data type, not through an extra bit here (for simplicity).
- For variable length fields: BigIntField representing the offset and length in the record is 0.
- For all fix length data types a fixed value is used.
 - `DataType.getNullValue()`
 - The serialization methods of `DataField` will write the correct values.

Record addressing

- Records are addressed through RIDs.
 - **Record IDentifiers** (also **Row IDentifiers**)
- RIDs are 64 bit integers, consisting of two parts.
 - Page number (more significant 32 bits)
 - Record position (less significant 32 bits)
- RIDs are a data type.
 - Can be part of a tuple during query execution.

Page expiration

- The data in `TablePage` is data that will be in the memory cache of the database system.
- The cache loads pages and throws pages out, overwriting the byte arrays with new data. Once a page is thrown out of the cache, it is marked as expired.
- Any operation on that page should then throw a *PageExpiredException* to indicate that situation.
- If the code is correct, that should never happen, but it does happen during later development, so we include an extra check here to catch potential bugs later.

Exercise 1

- Implement the `TablePage` interface.
- Implement the associated `TupleIterator` and `TupleRIDIterator` interfaces.
- Implement the factory methods `createTablePage()` and `initTablePage()`.

Tests and points

Test name	Points	Description
testInitializationOfNewPage*	0.5	Header information and TablePage metadata should be initialized correctly.
testInitializationOfOldPage*	0.5	Header information should correspond to binary data.
testRetrievalOfFullTuples	1	Fully stored tuples are retrieved correctly (no projection and no selection).
testRetrievalWithProjection	0.5	Returned tuples only contained projected values.
testRetrievalWithPredicates	0.5	Returned tuples satisfy predicates (no projection).
testRetrievalWithProjectionAndPredicates	0.5	Handle both projection and selection.
testExpirationOfEntries	0.5	Deleted tuples are not returned.
testExceptions	0.5	PageExpiredException is thrown when a page is expired.
testGetHeaderFields*	0.5	Header information of new page should be initialized correctly.
testIterator	1	Iterator should return all tuples of the page.
testIteratorWithRID*	0.5	Iterator should return all tuples of the page with their corresponding RID.
testIteratorWithPredicates	0.5	Iterator should return the tuples of the page that pass predicates.

* Only on the evaluation server.